programming assignment 4 - CUDA
due: Mon 4/11

using CUDA at Ohio Supercomputer Center

The OSC clusters are equipped with Tesla K40 GPUs. Some relevant stats for the K40:

```
  Total amount of global memory:                12288 MBytes (12884705280 bytes)
  (15) Multiprocessors, (192) CUDA Cores/MP:    2880 CUDA Cores
  GPU Clock rate:                               876 MHz (0.88 GHz)
  Memory Clock rate:                            3004 Mhz
  Memory Bus Width:                             384-bit
  L2 Cache Size:                                1572864 bytes
  Total amount of constant memory:              65536 bytes
  Total amount of shared memory per block:      49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                    32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:          1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
```

To use CUDA on the OSC cluster, you must allocate a node with an attached GPU. To interactively allocate such a node, use:

$ **qsub -I -l walltime=0:59:00 -l nodes=1:gpus=1**
(qsub -EYE -ell walltime ... -ell nodes ...).

To ensure the best resource availability for everyone, please only log on to a GPU host node when you are ready compile and run, then please exit when you are not actively testing.

To compile and test your programs you will need to load the CUDA environment:

$ **module load cuda**

and then use the Nvidia compilers. For example:

$ **nvcc -O -o lab4p1 jones_jeffrey_lab4p1.cu**

The "-O" flag sets the compiler to the default level (3), which the "-o lab4p1" flag, specifies the name for the the executable file, "lab4p1," which you can then execute by name:

$ **lab4p1**

Note that compilation (use the nvidia compiler, nvcc) can be performed on the login nodes, and does not require a node with a GPU. You will need to load the cuda module on the login node if you wish to do this. You will not be able to test your programs successfully on the login nodes, as they have no GPUs.

Nvidia CUDA drivers available free on-line

If your laptop/desktop has an Nvidia graphics card, you can download the CUDA drivers directly from Nvidia for your own local development and testing. Please see `https://developer.nvidia.com/cuda-downloads`. Nvidia's "CUDA Zone" also provides a wide array of tools and documentation: `https://developer.nvidia.com/cuda-zone`.

assignment 4

Part 1: Create both serial and CUDA parallel programs based on the following code segment:

```
double F[4097][4097];
// insert code to initialize array elements to random values between 1.0 and 2.0
for (k = 0; k < 100; k++)
    for (i = 1; i < 4097; i++)
        for (j=0; j < 4096; j++)
            F[ i ][ j ] = F[ i-1 ][ j+1 ] + F[ i ][ j+1 ];
```

Measure the performance of your **serial** version on OSC using 1 node with 12 processors (full node), and report your results in estimated GFlops. Then, measure the performance for the best CUDA parallel version you can engineer. Compare your results and explain your observations.

Part 2: Create both serial and CUDA parallel programs based on the following code segment, which multiplies the transpose of a matrix with itself:

```
double A[4096], C[4096];
// insert code to initialize matrix elements to random values between 1.0 and 2.0
for (i = 0; i < 4096; i++)
    for (j = 0; j < 4096; j++)
        for (k = 0; k < 4096; k++)
            C[ i ][ j ] += A[ k ][ i ] * A[ k ][ j ];
```

Use the code as above for your serial version. Use whatever techniques you feel appropriate to improve your parallel version. As before, measure both serial and parallel performance. Report and explain your results.

Part 3: Implement a CUDA program using a kernel to transpose the contents of a matrix in place (meaning, use only one matrix on the device). Use an integer matrix of size 1024 x 1024, initializing on the host to random values between 1 and 1000. Copy the device results back to the host, and verify that the transposition is correct. Run your program using:

- one block of 32 threads;
- one block of 1024 threads;
- two blocks of 1024 threads each.

Compare the performance of the three options. Did you get any unexpected results?

As a final experiment, run your program using 16 blocks of 1024 threads each. Describe what happens. What was the run time of this exercise. Did you get any unexpected results?

submitting results

Generally, follow the submission guidelines for the previous labs, with the following specifics:

- use submission directory name "cse5441_lab4."
- name your program files <lastname>_<firstname>_lab4p1, <lastname>_<firstname>_lab4p2 and <lastname>_<firstname>_lab4p3.
- provide a single make file that will name your executables lab4p1, lab4p2 and lab4p3.
- **submit a printed copy** of your report in class on Tuesday, 4/12. Be sure to include in your report your name and section number as well as all relevant CUDA parameter settings you used.